# Textbook Quality Dataset Method Review

According to the paper, “Textbooks are all you need”, a smaller model can achieve comparative higher accuracy if the dataset given to it is of more quality. They introduced the notion of ‘textbook quality dataset generation’ on which our methodology is mapped. According to the paper, the dataset provided to a model must possess several key characteristics to maximize its effectiveness:

* **Quality/Educational Value:** The dataset should be carefully curated to ensure that the content has high educational value. This means that the examples should be clear, instructive, and aligned with the learning objectives of the model.
* **Diversity/Non-Repetitiveness:** To avoid redundancy and promote a broad understanding, the dataset must be diverse. This diversity is achieved by varying the topics, ensuring that the content is non-repetitive both in text and code. The dataset should include examples that differ in terms of difficulty, complexity, and style, allowing the model to generalize across a wide range of scenarios.
* **Difficulty:** The dataset should encompass a range of difficulties, from simple to challenging problems. This ensures that the model can handle both basic tasks and more complex ones, leading to better performance across different levels of difficulty.
* **Complexity:** Similarly, the complexity of the examples should vary, with some problems requiring simple solutions and others demanding more sophisticated approaches. This helps the model develop a deeper understanding of the subject matter.
* **Style:** The style of the examples should also be varied, incorporating different coding styles, comment structures, and approaches to problem-solving. This stylistic diversity further enhances the model's ability to adapt to various programming paradigms and real-world coding practices.

The authors also highlight the importance of generating data that encourages reasoning and the development of basic algorithmic skills.

# Employed Technique

Similar to what the authors have done, our methodology entails using GPT-4 to help create a higher-quality dataset from the one we have created in previous labs.

1. **Generating Quality Dataset:**

Following in the prompt used for the “Multivariate” equations. (Have employed this to all data generations)

|  |
| --- |
|  |
| Output code is attached in file alongside |

We will provide a similar prompt for each equation-type. Hence generating diverse data. GPT-4 improves each code to be more diverse:

A black screen with white text

Description automatically generated

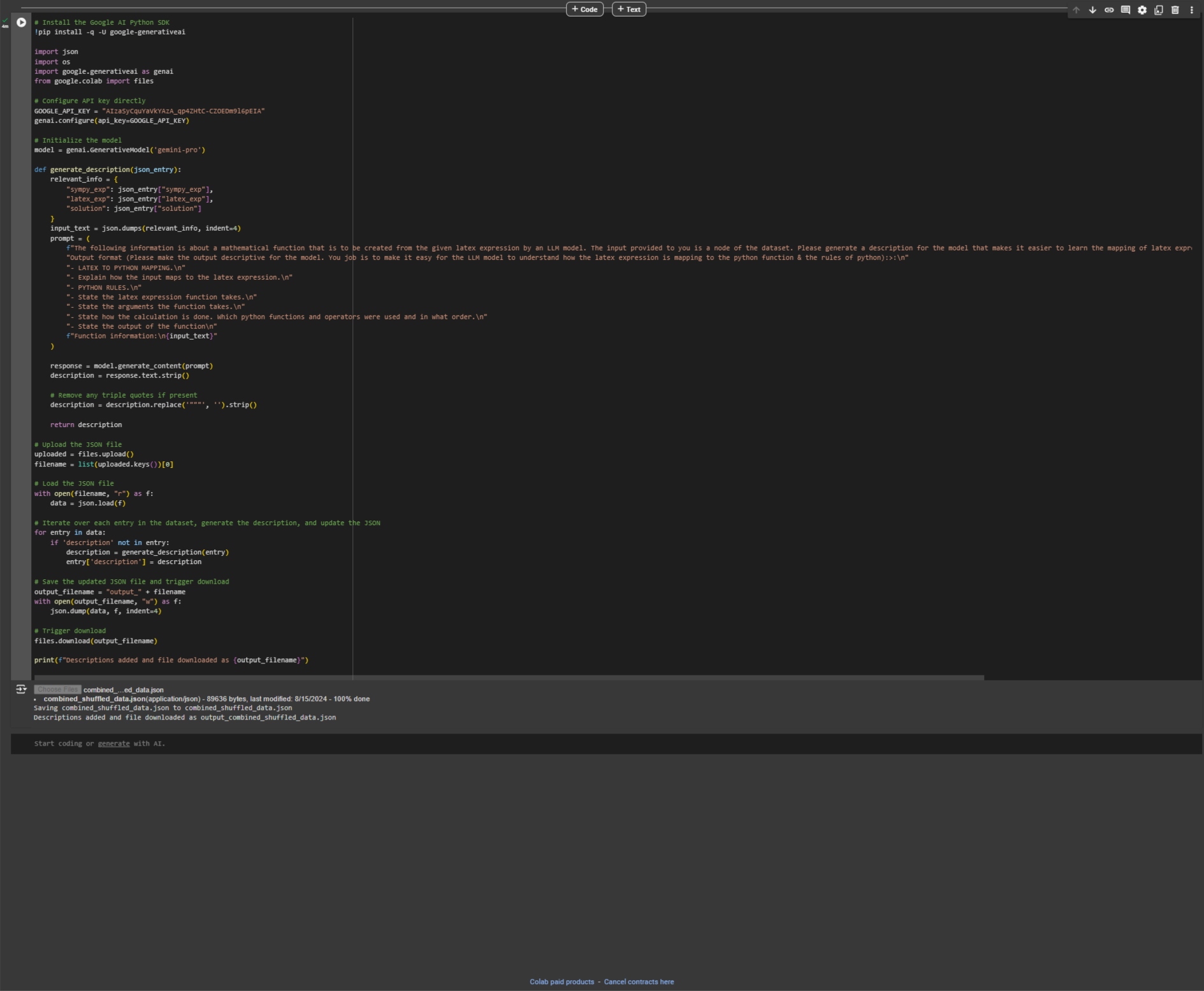
1. **Adding description of Function:**

For each json, we added another key that gives the model more information about the function and the task, similar to how a child would be given context of every topic he/she reads in a textbook. Following is the test chat we had with GPT-4:

|  |
| --- |
|  |
| {  "task\_id": "1",  "sympy\_exp": "-2.64155391148627\*v\*y\*\*2\*z\*\*2 + 3.82838915724899\*w\*\*6 - 0.230104152676097\*w\*\*4\*x\*\*3 - 0.141689971281759\*w\*\*4\*y\*\*4\*z\*\*4 - 1.62967717772171\*w\*y\*\*3",  "latex\_exp": "- 2.64155391148627 v y^{2} z^{2} + 3.82838915724899 w^{6} - 0.230104152676097 w^{4} x^{3} - 0.141689971281759 w^{4} y^{4} z^{4} - 1.62967717772171 w y^{3}",  "solution": "def \_lambdifygenerated(x, y, z, w, v):\n return -2.64155391148627\*v\*y\*\*2\*z\*\*2 + 3.82838915724899\*w\*\*6 - 0.230104152676097\*w\*\*4\*x\*\*3 - 0.141689971281759\*w\*\*4\*y\*\*4\*z\*\*4 - 1.62967717772171\*w\*y\*\*3",  "simplified\_solution": "-2.64155391148627\*v\*y\*\*2\*z\*\*2 + 3.82838915724899\*w\*\*6 - 0.230104152676097\*w\*\*4\*x\*\*3 - 0.141689971281759\*w\*\*4\*y\*\*4\*z\*\*4 - 1.62967717772171\*w\*y\*\*3",  "synthetic": true,  "domain": "Mathematics\_Algebra",  "test\_cases": [  {  "input": {  "x": -5.483643360697541,  "y": -3.2038360480124055,  "z": -5.674140585154134,  "w": -7.785306644243932,  "v": -7.969417099324874  },  "output": -55850704.6997375  },  {  "input": {  "x": 8.095056911182482,  "y": -9.907773647448,  "z": -7.539085240677128,  "w": -1.112703522278533,  "v": 1.944903962799211  },  "output": -6791996.543856193  },  {  "input": {  "x": -6.92783317246203,  "y": -5.30080948122559,  "z": -6.425107760487219,  "w": 9.51240412234397,  "v": -0.012690884804767322  },  "output": -1557491114.000856  },  {  "input": {  "x": 9.159930580845902,  "y": 2.946761569944851,  "z": 6.4763648987263664,  "w": 0.9106964820256795,  "v": -8.05519572719988  },  "output": -5335.905834457512  },  {  "input": {  "x": -6.204460033963208,  "y": -7.765296528371031,  "z": 3.0317613490405684,  "w": 8.613026640129405,  "v": 2.9557011880293533  },  "output": -237670374.74626616  }  ],  "complexity": "8",  "equation\_type": "algebraic",  "output\_type": "real",  **"function\_description": "This function, named `\_lambdifygenerated`, takes five arguments (`x`, `y`, `z`, `w`, `v`) and evaluates a complex multivariate algebraic expression involving these variables. The function is generated by converting a SymPy expression into executable Python code using the `lambdify` method. The output of this function is a real number, computed based on the input values for the variables `x`, `y`, `z`, `w`, and `v`, following the operations defined by the original SymPy expression."**  } |

Now, since GPT-4 has a paid API call, we use Gemini for creating descriptions:

|  |
| --- |
| # Install the Google AI Python SDK  !pip install -q -U google-generativeai  import json  import os  import google.generativeai as genai  from google.colab import files  # Configure API key directly  GOOGLE\_API\_KEY = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"  genai.configure(api\_key=GOOGLE\_API\_KEY)  # Initialize the model  model = genai.GenerativeModel('gemini-pro')  def generate\_description(json\_entry):      relevant\_info = {          "sympy\_exp": json\_entry["sympy\_exp"],          "latex\_exp": json\_entry["latex\_exp"],          "solution": json\_entry["solution"]      }      input\_text = json.dumps(relevant\_info, indent=4)      prompt = (          f"The following information is about a mathematical function that is to be created from the given latex expression by an LLM model. The input provided to you is a node of the dataset. Please generate a description for the model that makes it easier to learn the mapping of latex expression to python code and the rules of the python code."          "Output format (Please make the output descriptive for the model. You job is to make it easy for the LLM model to understand how the latex expression is mapping to the python function & the rules of python):>:\n"          "- LATEX TO PYTHON MAPPING.\n"          "- Explain how the input maps to the latex expression.\n"          "- PYTHON RULES.\n"          "- State the latex expression function takes.\n"          "- State the arguments the function takes.\n"          "- State how the calculation is done. Which python functions and operators were used and in what order.\n"          "- State the output of the function\n"          f"Function information:\n{input\_text}"      )        response = model.generate\_content(prompt)      description = response.text.strip()        # Remove any triple quotes if present      description = description.replace('"""', '').strip()        return description  # Upload the JSON file  uploaded = files.upload()  filename = list(uploaded.keys())[0]  # Load the JSON file  with open(filename, "r") as f:      data = json.load(f)  # Iterate over each entry in the dataset, generate the description, and update the JSON  for entry in data:      if 'description' not in entry:          description = generate\_description(entry)          entry['description'] = description  # Save the updated JSON file and trigger download  output\_filename = "output\_" + filename  with open(output\_filename, "w") as f:      json.dump(data, f, indent=4)  # Trigger download  files.download(output\_filename)  print(f"Descriptions added and file downloaded as {output\_filename}") |
| "description": "\*\*LATEX TO PYTHON MAPPING\*\*\n\nThe input is `x z + x - y^3 - z^2` in LaTeX format.\n\n\*\*PYTHON RULES\*\*\n\nThe corresponding Python function takes three arguments: `x`, `y`, and `z`.  The exponential operator in LaTeX is represented by `\*\*` in Python. This means that `y^3` becomes `y\*\*3` in Python.\n\n\*\*PYTHON FUNCTION\*\*\n\n```\ndef \_lambdifygenerated(x, y, z):\n    return x\*z + x - y\*\*3 - z\*\*2\n```\n\nThis function takes three arguments, `x`, `y`, and `z`, and returns the value of the expression `x z + x - y^3 - z^2`.\n\n\*\*OUTPUT\*\*\n\nThe output of the function is a single numerical value, which is the result of evaluating the expression for the given values of `x`, `y`, and `z`." |



And this is how we generate docstring for the dataset